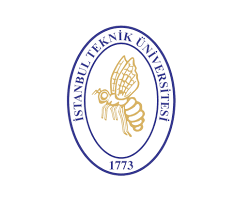
EHB436E Digital System Design Applications



Final Project Report

Mehmet Yasir Bağcı

040200037

Salih Ömer Ongün

040220780

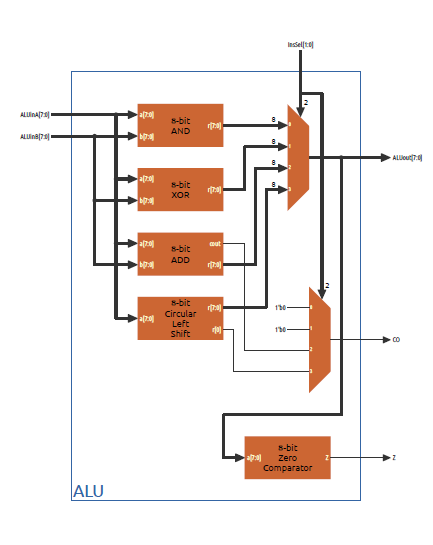
1. **Problem Statement**

The problem that was assigned to us is calculating the by using the blocks that are given to us. The blocks that we utilized are:

* ALU
* Register Block
* Control Unit
* Top module

We implement our algorithm by inserting the relative code to control unit block.

1. **ALU**

****

Şekil ALU Block Diagram

**module** ALU

**(**

**input** **[**1**:**0**]** insel**,**

**output** **[**7**:**0**]** alu\_out**,**

**input** **[**7**:**0**]** alu\_in\_a**,**

**input** **[**7**:**0**]** alu\_in\_b**,**

**output** co**,**

**output** z

**);**

**wire** cout**;**

**wire** **[**7**:**0**]** add\_sum**,**shift\_out**,**and\_out**,**xor\_out**;**

**wire** shift\_out\_0**;**

// ALU\_OUT\_MUX modülünün örneklemesi

alu\_out\_mux alu\_mux\_inst **(**

**.**D0**(**and\_out**),**

**.**D1**(**xor\_out**),**

**.**D2**(**add\_sum**),**

**.**D3**(**shift\_out**),**

**.**insel**(**insel**),**

**.**O**(**alu\_out**)**

**);**

// CO\_MUX modülünün örneklemesi

co\_mux co\_mux\_inst **(**

**.**D0**(**1'b0**),**

**.**D1**(**1'b0**),**

**.**D2**(**cout**),**

**.**D3**(**shift\_out\_0**),**

**.**insel**(**insel**),**

**.**O**(**co**)**

**);**

AND and\_8

**(**

**.**a**(**alu\_in\_a**),**

**.**b**(**alu\_in\_b**),**

**.**r**(**and\_out**)**

**);**

XOR xor\_8

**(**

**.**a**(**alu\_in\_a**),**

**.**b**(**alu\_in\_b**),**

**.**r**(**xor\_out**)**

**);**

ADD add\_8

**(**

**.**x**(**alu\_in\_a**),**

**.**y**(**alu\_in\_b**),**

**.**cin**(**1'b0**),**

**.**cout**(**cout**),**

**.**sum**(**add\_sum**)**

**);**

circular\_shift shift8

**(**

**.**a**(**alu\_in\_a**),**

**.**r**(**shift\_out**),**

**.**r\_0**(**shift\_out\_0**)**

**);**

zero\_comp comparator8

**(**

**.**a**(**alu\_out**),**

**.**z**(**z**)**

**);**

**endmodule**

**module** alu\_out\_mux **(**

**input** **[**7**:**0**]** D0**,**

**input** **[**7**:**0**]** D1**,**

**input** **[**7**:**0**]** D2**,**

**input** **[**7**:**0**]** D3**,**

**input** **[**1**:**0**]** insel**,** // 2-bit seçim girişi

**output** **reg** **[**7**:**0**]** O // 4-bit çıkış

**);**

always @(\*)

begin

case (insel)

2'b00: O = D0;

2'b01: O = D1;

2'b10: O = D2;

2'b11: O = D3;

default: O = 8'b0; // Güvenlik için bir default durumu ekledik

endcase

end

endmodule

**module** co\_mux **(**

**input** D0**,**

**input** D1**,**

**input** D2**,**

**input** D3**,**

**input** **[**1**:**0**]** insel**,** // 2-bit seçim girişi

**output** **reg** O // 4-bit çıkış

**);**

**always** **@(\*)**

**begin**

**case** **(**insel**)**

2'b00**:** O **=** D0**;**

2'b01**:** O **=** D1**;**

2'b10**:** O **=** D2**;**

2'b11**:** O **=** D3**;**

**default:** O **=** 0**;** // Güvenlik için bir default durumu ekledik

**endcase**

**end**

**endmodule**

**module** circular\_shift

**(**

**input** **[**7**:**0**]** a **,**

**output** **[**7**:**0**]** r**,**

**output** r\_0

**);**

**assign** r **=** **{**a**[**6**:**0**],**a**[**7**]};**

**assign** r\_0 **=** r**[**0**];**

**endmodule**

**module** XOR**(**

**input** **[**7**:**0**]** a**,**

**input** **[**7**:**0**]** b**,**

**output** **[**7**:**0**]** r**);**

**assign** r **=** a **^** b**;**

**endmodule**

**module** AND**(**

**input** **[**7**:**0**]** a**,**

**input** **[**7**:**0**]** b**,**

**output** **[**7**:**0**]** r

**);**

**assign** r **=** a **&** b**;**

**endmodule**

**module** ADD

**(**

**input** **[**7**:**0**]** x**,**

**input** **[**7**:**0**]** y**,**

**input** cin**,**

**output** cout**,**

**output** **[**7**:**0**]** sum

**);**

**wire** **[**8**:**0**]** cout\_gen**;**

**assign** cout\_gen**[**0**]** **=** cin**;**

**genvar** i**;**

**generate**

**for(**i **=** 0**;** i**<**8**;** i **=** i**+**1**)** **begin** **:** gen\_full\_adder

FA gen\_full

**(**

x**[**i**],**

y**[**i**],**

cout\_gen**[**i**],**

cout\_gen**[**i**+**1**],**

sum**[**i**]**

**);**

**end**

**endgenerate**

**assign** cout **=** cout\_gen**[**8**];**

**endmodule**

**module** HA

**(**

**input** x**,**

**input** y**,**

**output** cout**,**

**output** sum

**);**

**reg** sum\_reg**,**cout\_reg**;**

**always@(**x**,**y**)** **begin**

**{**cout\_reg**,**sum\_reg**}** **=** x **+** y**;**

**end**

**assign** sum **=** sum\_reg**;**

**assign** cout **=** cout\_reg**;**

**endmodule**

**module** FA

**(**

**input** x**,**

**input** y**,**

**input** cin**,**

**output** cout**,**

**output** sum

**);**

**wire** ha1\_sum**,** ha1\_cout**,** ha2\_cout**;**

HA half1

**(**

**.**x**(**x**),**

**.**y**(**y**),**

**.**sum**(**ha1\_sum**),**

**.**cout**(**ha1\_cout**)**

**);**

HA half2

**(**

**.**x**(**ha1\_sum**),**

**.**y**(**cin**),**

**.**sum**(**sum**),**

**.**cout**(**ha2\_cout**)**

**);**

OR or1

**(**

**.**l1**(**ha1\_cout**),**

**.**l2**(**ha2\_cout**),**

**.**O**(**cout**)**

**);**

**endmodule**

**module** OR

**(**

**input** l1**,**

**input** l2**,**

**output** O

**);**

**assign** O **=** l1 **|** l2**;**

**endmodule**

Şekil Verilog Code of ALU Block and Inner modules

**module** zero\_comp

**(**

**input** **[**7**:**0**]** a**,**

**output** **reg** z

**);**

**always** **@(\*)** **begin**

**if** **(**a **==** 8'b0**)** **begin**

z **<=** 1**;**

**end**

**else** **begin**

z **<=** 0**;**

**end**

**end**

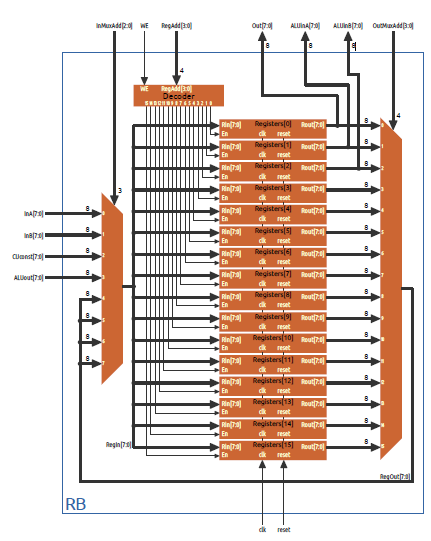
**endmodule**

diyagram, plan, metin, teknik çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şekil RTL Schematic

1. **Register Block**



Şekil RB Block Diagram

**module** REGISTER\_BLOCK

**(**

**input** clk**,**

**input** rst**,**

**input** **[**7**:**0**]** inA**,**

**input** **[**7**:**0**]** inB**,**

**input** **[**7**:**0**]** cu\_const**,**

**input** **[**2**:**0**]** in\_mux\_add**,**

**input** **[**3**:**0**]** out\_mux\_add**,**

**input** **[**3**:**0**]** reg\_add**,**

**input** we**,**

**input** **[**7**:**0**]** alu\_out**,**

**output** **[**7**:**0**]** alu\_in\_a**,**

**output** **[**7**:**0**]** alu\_in\_b**,**

**output** **[**7**:**0**]** out

**);**

**wire** **[**15**:**0**]** decoder\_out**;**

**wire** **[**7**:**0**]** out\_mux\_out**;**

**wire** **[**7**:**0**]** in\_mux\_out**;**

**wire** **[**7**:**0**]** re\_out0**,** re\_out1**,** re\_out2**,** re\_out3**,** re\_out4**,** re\_out5**,**re\_out6**;**

// DECODER instantiation

DECODER decoder

**(**

**.**reg\_add**(**reg\_add**),**

**.**we**(**we**),**

**.**OUT**(**decoder\_out**)**

**);**

// IN\_MUX instantiation

IN\_MUX u\_in\_mux

**(**

**.**D0**(**inA**),**

**.**D1**(**inB**),**

**.**D2**(**cu\_const**),**

**.**D3**(**alu\_out**),**

**.**D4**(**out\_mux\_out**),**

**.**D5**(**out\_mux\_out**),**

**.**D6**(**out\_mux\_out**),**

**.**D7**(**out\_mux\_out**),**

**.**in\_mux\_add**(**in\_mux\_add**),**

**.**O**(**in\_mux\_out**)**

**);**

// OUT\_MUX instantiation

OUT\_MUX u\_out\_mux

**(**

**.**D0**(**re\_out0**),**

**.**D1**(**re\_out1**),**

**.**D2**(**re\_out2**),**

**.**D3**(**re\_out3**),**

**.**D4**(**re\_out4**),**

**.**D5**(**re\_out5**),**

**.**D6**(**re\_out6**),**

**.**D7**(**8'b0**),**

**.**D8**(**8'b0**),**

**.**D9**(**8'b0**),**

**.**D10**(**8'b0**),**

**.**D11**(**8'b0**),**

**.**D12**(**8'b0**),**

**.**D13**(**8'b0**),**

**.**D14**(**8'b0**),**

**.**D15**(**8'b0**),**

**.**out\_mux\_add**(**out\_mux\_add**),**

**.**O**(**out\_mux\_out**)**

**);**

// REGISTER instantiations

REGISTER reg0

**(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**rin**(**in\_mux\_out**),**

**.**en**(**decoder\_out**[**0**]),**

**.**rout**(**re\_out0**)**

**);**

REGISTER reg1

**(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**rin**(**in\_mux\_out**),**

**.**en**(**decoder\_out**[**1**]),**

**.**rout**(**re\_out1**)**

**);**

REGISTER reg2

**(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**rin**(**in\_mux\_out**),**

**.**en**(**decoder\_out**[**2**]),**

**.**rout**(**re\_out2**)**

**);**

REGISTER reg3

(

.clk(clk),

.rst(rst),

.rin(in\_mux\_out),

.en(decoder\_out[3]),

.rout(re\_out3)

);

REGISTER reg4

(

.clk(clk),

.rst(rst),

.rin(in\_mux\_out),

.en(decoder\_out[4]),

.rout(re\_out4)

);

REGISTER reg5

(

.clk(clk),

.rst(rst),

.rin(in\_mux\_out),

.en(decoder\_out[5]),

.rout(re\_out5)

);

REGISTER reg6

**(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**rin**(**in\_mux\_out**),**

**.**en**(**decoder\_out**[**6**]),**

**.**rout**(**re\_out6**)**

**);**

// Output assignment

**assign** alu\_in\_a **=** re\_out1**;**

**assign** alu\_in\_b **=** re\_out2**;**

**assign** out **=** re\_out0**;**

**endmodule**

**module** DECODER

**(**

**input** **[**3**:**0**]** reg\_add**,**

**input** we**,** // write enable sinyali

**output** **[**15**:**0**]** OUT

**);**

reg [15:0] out\_reg;

always @(\*)

begin

if (we) begin // we sinyali 1 ise çalışır

case(reg\_add)

4'b0000: out\_reg = 16'b0000\_0000\_0000\_0001;

4'b0001: out\_reg = 16'b0000\_0000\_0000\_0010;

4'b0010: out\_reg = 16'b0000\_0000\_0000\_0100;

4'b0011: out\_reg = 16'b0000\_0000\_0000\_1000;

4'b0100: out\_reg = 16'b0000\_0000\_0001\_0000;

4'b0101: out\_reg = 16'b0000\_0000\_0010\_0000;

4'b0110: out\_reg = 16'b0000\_0000\_0100\_0000;

4'b0111**:** out\_reg **=** 16'b0000\_0000\_1000\_0000**;**

4'b1000**:** out\_reg **=** 16'b0000\_0001\_0000\_0000**;**

4'b1001**:** out\_reg **=** 16'b0000\_0010\_0000\_0000**;**

4'b1010**:** out\_reg **=** 16'b0000\_0100\_0000\_0000**;**

4'b1011**:** out\_reg **=** 16'b0000\_1000\_0000\_0000**;**

4'b1100**:** out\_reg **=** 16'b0001\_0000\_0000\_0000**;**

4'b1101**:** out\_reg **=** 16'b0010\_0000\_0000\_0000**;**

4'b1110**:** out\_reg **=** 16'b0100\_0000\_0000\_0000**;**

4'b1111**:** out\_reg **=** 16'b1000\_0000\_0000\_0000**;**

**default:** out\_reg **=** 16'b0000\_0000\_0000\_0000**;**

**endcase**

**end** **else** **begin** // we sinyali 0 ise çıkış sıfırlanır

out\_reg **=** 16'b0000\_0000\_0000\_0000**;**

**end**

**end**

assign OUT = out\_reg;

endmodule

module IN\_MUX

(

input [7:0] D0,

input [7:0] D1,

input [7:0] D2,

input [7:0] D3,

input [7:0] D4,

input [7:0] D5,

input [7:0] D6,

input [7:0] D7,

input [2:0] in\_mux\_add, // 3-bit seçim girişi

output reg [7:0] O // 8-bit çıkış

);

always @(\*)

begin

case (in\_mux\_add)

3'b000: O = D0;

3'b001: O = D1;

3'b010: O = D2;

3'b011: O = D3;

3'b100: O = D4;

3'b101: O = D5;

3'b110: O = D6;

3'b111: O = D7;

default: O = 8'b0; // için bir default durumu ekledik

endcase Güvenlik

end

endmodule

**module** OUT\_MUX

**(**

**input** **[**7**:**0**]** D0**,**

**input** **[**7**:**0**]** D1**,**

**input** **[**7**:**0**]** D2**,**

**input** **[**7**:**0**]** D3**,**

**input** **[**7**:**0**]** D4**,**

**input** **[**7**:**0**]** D5**,**

**input** **[**7**:**0**]** D6**,**

**input** **[**7**:**0**]** D7**,**

**input** **[**7**:**0**]** D8**,**

**input** **[**7**:**0**]** D9**,**

**input** **[**7**:**0**]** D10**,**

**input** **[**7**:**0**]** D11**,**

**input** **[**7**:**0**]** D12**,**

**input** **[**7**:**0**]** D13**,**

**input** **[**7**:**0**]** D14**,**

**input** **[**7**:**0**]** D15**,**

**input** **[**3**:**0**]** out\_mux\_add**,** // 4-bit seçim girişi

**output** **reg** **[**7**:**0**]** O // 8-bit çıkış

**);**

always @(\*)

begin

case (out\_mux\_add)

4'b0000: O = D0;

4'b0001: O = D1;

4'b0010: O = D2;

4'b0011: O = D3;

4'b0100: O = D4;

4'b0101: O = D5;

4'b0110: O = D6;

4'b0111: O = D7;

4'b1000: O = D8;

4'b1001: O = D9;

4'b1010: O = D10;

4'b1011: O = D11;

4'b1100: O = D12;

4'b1101: O = D13;

4'b1110: O = D14;

4'b1111: O = D15;

default: O = 8'b0; // Default durumu, güvenlik için

endcase

end

endmodule

**module** REGISTER

**(**

**input** clk**,**

**input** rst**,**

**input** **[**7**:**0**]** rin**,**

**input** en**,**

**output** **reg** **[**7**:**0**]** rout

**);**

Şekil Verilog Code of Register Block

**if** **(**rst **==** 1'b1**)** **begin**

rout **<=** 8'b0**;**

**end**

**else** **begin**

**if** **(**en **==** 1'b1**)** **begin**

rout **<=** rin**;**

**end**

**end**

**end**

**endmodule**

diyagram, plan, metin, teknik çizim içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şekil Register Block RTL Schematic

1. **Control Unit**

**EXPONENTIAL CALCULATING**

Register4 holds the sum value of the result of each operation. When decreasing the upper, we assign the total value (Register4) to Base (Register6).

**EXAMPLE**

**Firstly**

**FIRST LOOP**

**4 + 4 = 8**

**8 + 4 = 12**

**12+ 4 = 16**

**SECOND LOOP**

**16 + 16 = 32**

**32 + 16 = 48**

**48 + 16 = 64**

**THIRD LOOP**

**Go to DONE state**

**OUT <=**

**State goes to DONE and Register4 assigned to OUT.**

**module** control\_unit

**(**

**input** clk**,**

**input** rst**,**

**input** start**,**

**input** co**,**

**input** z**,**

**output** **reg** busy**,**

**output** **reg** **[**1**:**0**]** insel**,**

**output** **[**7**:**0**]** cu\_const**,**

**output** **reg** **[**2**:**0**]** in\_mux\_add**,**

**output** **reg** **[**3**:**0**]** out\_mux\_add**,**

**output** **reg** **[**3**:**0**]** reg\_add**,**

**output** **reg** we

**);**

**localparam** IDLE **=** 5'b00000**;**

**localparam** START **=** 5'b00001**;**

**localparam** ZERO\_A **=** 5'b00010**;**

**localparam** ZERO\_B **=** 5'b00011**;**

**localparam** OUT1 **=** 5'b00100**;**

**localparam** OUT1\_1 **=** 5'b00101**;**

**localparam** OUT1\_2 **=** 5'b00110**;**

**localparam** EXP\_SUB **=** 5'b00111**;**

**localparam** SUB\_REC **=** 5'b01000**;**

**localparam** SEL\_BASE **=** 5'b01001**;**

**localparam** EXP\_TO\_BASE **=** 5'b01010**;**

**localparam** MULT\_SUB **=** 5'b01011**;**

**localparam** ADD\_REC **=** 5'b01100**;**

**localparam** MULT\_ADD **=** 5'b01101**;**

**localparam** SEL\_SUM\_A **=** 5'b01110**;**

**localparam** ADD\_RES **=** 5'b01111**;**

**localparam** SEL\_A **=** 5'b10000**;**

**localparam** DONE **=** 5'b10001**;**

**localparam** ARA1 **=** 5'b10010**;**

**localparam** ARA2 **=** 5'b10011**;**

**reg** **[**4**:**0**]** state**;**

**assign** cu\_const **=** 8'b11111111**;**

**always** **@(posedge** clk **or** **posedge** rst**)** **begin**

**if** **(**rst **==** 1'b1**)** **begin**

state **<=** IDLE**;**

we **<=** 0**;**

**end**

**else** **begin**

**case(**state**)**

IDLE**:** **begin**

**if(**start **==** 1'b1**)** **begin**

state **<=** START**;**

we **<=** 1'b1**;**

busy **<=** 1**;**

in\_mux\_add **<=** 3'd0**;**

reg\_add **<=** 4'd4**;** // A degerini registira atadık hem B = 1 hem de sonraki işlemde ekleme yakmak icin

**end**

**end**

START**:** **begin**

in\_mux\_add **<=** 3'd0**;** // A secildi

reg\_add **<=** 4'd1**;** // A inA ya verildi

state **<=** ARA1**;**

**end**

ARA1**:** **begin**

insel **<=** 2'd3**;** // shift yapildi

state **<=** ZERO\_A**;**

**end**

ZERO\_A**:** **begin** // A sıfır mı

**if(** z **==** 1**)** **begin**

state **<=** DONE**;**

**end**

**else** **begin**

state **<=** ARA2**;**

in\_mux\_add **<=** 3'd1**;** // B seciyorum

reg\_add **<=** 4'd1**;** // B yi inA veriyorum

**end**

**end**

ARA2**:** **begin**

insel **<=** 2'd3**;** // shift yapildi

state **<=** ZERO\_B**;**

**end**

ZERO\_B**:** **begin**

**if(** z **==** 1**)** **begin**

state **<=** OUT1**;**

in\_mux\_add **<=** 3'd2**;** // cuconst secildi

reg\_add **<=** 4'd1**;** // inA ya verildi

**end**

**else** **begin**

state **<=** EXP\_SUB**;**

in\_mux\_add **<=** 3'd1**;** // B seciyorum

reg\_add **<=** 4'd1**;** // B yi inA veriyorum

**end**

**end**

OUT1**:** **begin**

state **<=** OUT1\_1**;**

in\_mux\_add **<=** 3'd2**;** // cuconst secildi

reg\_add **<=** 4'd2**;** // inB ye verildi

**end**

OUT1\_1**:** **begin**

state **<=** OUT1\_2**;**

insel **<=** 2'd2**;** // 255 + 255 yapilip 510 yani 1\_1111\_1110 bulundu

in\_mux\_add **<=** 3'd3**;** // ALUout (1\_1111\_1110) secildi

reg\_add **<=** 4'd1**;** // inA ye verildi

**end**

OUT1\_2**:** **begin**

state **<=** DONE**;**

insel **<=** 2'd1**;** // 255 ile 510 exor oldu 0000\_0001 yani decimal 1 bulundu

in\_mux\_add **<=** 3'd3**;** // ALUout (0000\_0001) secildi

reg\_add **<=** 4'd4**;** // DONE da outa verilen register4 e verildi

**end**

EXP\_SUB **:** **begin**

in\_mux\_add **<=** 3'd2**;** // constant sec

reg\_add **<=** 4'd2**;** // constant inB ye gonder

insel **<=** 2'd2**;** // B dan 1 cikar

state **<=** SUB\_REC**;**

**end**

SUB\_REC **:** **begin**

in\_mux\_add **<=** 3'd3**;** // Eksilen B yi (alu out) SEC

reg\_add **<=** 4'd3**;** // Register a at

**if** **(** z **==** 1**)** **begin** // ustte ifade kalmadı B = 0 oldu

state **<=** DONE**;**

//out\_mux\_add <= 4'd4; // toplam degeri feedback olarak verildi

**end**

**else** **begin**

state **<=** SEL\_BASE**;**

out\_mux\_add **<=** 4'd4**;** // toplam degeri feedback olarak verildi (base olarak kullanilacak)

**end**

**end**

SEL\_BASE**:** **begin**

out\_mux\_add **<=** 4'd4**;** // toplam degeri feedback olarak verildi (base olarak kullanilacak)

in\_mux\_add **<=** 3'd4**;** // base'i sec

reg\_add **<=** 4'd6**;** // base i registira atadik

state **<=** EXP\_TO\_BASE**;**

**end**

EXP\_TO\_BASE **:** **begin**

in\_mux\_add **<=** 3'd0**;** // A sec

reg\_add **<=** 4'd1**;** // A yi inA ya gonder

state **<=** MULT\_SUB**;**

**end**

MULT\_SUB**:** **begin**

in\_mux\_add **<=** 3'd2**;** // constant sec

reg\_add **<=** 4'd2**;** // constan i inB ya gonder

insel **<=** 2'd2**;** // A dan 1 cikar

state **<=** ADD\_REC**;**

**end**

ADD\_REC**:** **begin**

in\_mux\_add **<=** 3'd3**;** // Eksilen A yi (alu out) SEC

reg\_add **<=** 4'd5**;** // Register a at

state **<=** MULT\_ADD**;**

**end**

MULT\_ADD **:** **begin**

**if** **(**z **==** 1**)** **begin** // toplama bitti

state **<=** EXP\_SUB**;**

out\_mux\_add **<=** 4'd3**;** // register da tutulan B yi seciyorum

in\_mux\_add **<=** 3'd4**;** // feedback olan değeri seçtim

reg\_add **<=** 4'd1**;** // inA ya atiyorum

**end**

**else** **begin**

out\_mux\_add **<=** 4'd6**;** // register da tutulan base degerini sec

in\_mux\_add **<=** 3'd4**;** // base degerini sec

reg\_add **<=** 4'd1**;** // inA ya ver

state **<=** SEL\_SUM\_A**;**

**end**

**end**

SEL\_SUM\_A**:** **begin**

out\_mux\_add **<=** 4'd4**;** // register da tutulan degeri feedback yap

in\_mux\_add **<=** 3'd4**;** // registerda tutulan degeri sec

reg\_add **<=** 4'd2**;** // degeri inB ye ver

insel **<=** 2'd2**;** // base ile tutulan degeri topla

state **<=** ADD\_RES**;**

**end**

ADD\_RES **:** **begin**

in\_mux\_add **<=** 3'd3**;** // (toplanan sayiyi yani ALUout sec)

reg\_add **<=** 4'd4**;** // toplamı baska registire ata

state **<=** SEL\_A**;**

**end**

Şekil Control Unit Verilog Code

SEL\_A**:** **begin**

out\_mux\_add **<=** 4'd5**;** // Eksilen A yi sec

in\_mux\_add **<=** 3'd4**;**

reg\_add **<=** 4'd1**;** // eksilen A yi inA ya ver

state **<=** MULT\_SUB**;**

**end**

DONE**:** **begin**

out\_mux\_add **<=** 4'd4**;** // toplam degeri feedback olarak verildi

in\_mux\_add **<=** 4'd4**;** // son deger secildi

reg\_add **<=** 4'd0**;** // cikisa verildi

busy **<=** 0**;** // we yi sıfır yapınca cikisa deger gitmiyor

state **<=** IDLE**;**

**end**

**default:** **begin**

state **<=** IDLE**;**

we **<=** 0**;**

**end**

**endcase**

**end**

**end**

**endmodule**

We enable the registers in IDLE state. We give Input A to register4 to use it when collecting in the next states.

In START state we give input A to ALUinA. To check if A = 0.

Since the assigned value goes to ALUinA 1 clk later due to the register, we created the ARA1 state. When input A value comes to ALUinA, we make a circular shift. We check whether the value is zero at the output. If the value is zero, z = 1 and we understand that the input value is 0.

In the case of ZERO\_A, if z = 1, it goes to the DONE state (In the DONE state, the value in register4 is selected and given to OUT. We initially assigned the input A value to register4). If z = 0, this time input B value is assigned to ALUinA to check if B = 0.

In ARA2, B value is shifted and z output is obtained.

In ZERO\_B, if z = 1, i.e. B = 0, it goes to OUT1 state. If B != 0, it will continue for the addition.

We are trying to give 1 to OUT output at OUT1, OUT1\_1 and OUT1\_2. Because if the exponent of the number is 0, the result is 1. Cuconstant = 255 (8'b111111\_111111). To get the result 1, both ALUinA and ALUinB are given CUconstant value and summed. 9'b1\_111111\_1110 is obtained. Since we can get 8 bits, we get 8'b111111\_1110. When we XOR this number with cuconstant (8‘b111111\_1111) value, we get 8’b0000\_0001 (decimal 1) value in ALUout. OUT1\_2 is given to register4 for output and goes to DONE state.

If A!=0 or B!=0, it continues with EXP\_SUB status for exponentiation.

Firstly, we subtract 1 from the exponent number (B), in the EXP\_SUB state. For the subtraction process, I add the input B with the cuconst value of 8‘b1111\_111111. So I subtract 1 from B. I assigned the cuconst value 8’b111111\_1111 to make the subtraction process. When the subtraction process is done, we switch to SUB\_REC state.

In the SUB\_REC state, I assigned the decreasing result of B to register3 for the process to continue. The purpose here is to check whether z = 1 in SUB\_REC to check whether the result is zero when I subtract 1 from B. If it is zero, I switch to DONE state and output the result stored in the register. If Input B = 1, input A, which I assigned to register4 in START state, will be output when I subtract 1 from B. If B is greater than 1, the process will continue, register4 will be updated, and when the last B is zero, the updated result will be given to OUT in the DONE state. If it is greater than 1, it will switch to SEL\_BASE state and the sum value held in register4 (initially input A value) will be feedback.

In SEL\_BASE state, we assigned the total value to Register6 to use it as a base.

Register4 keeps the updated sum result at each operation in the exponentiation process. If the exponent is over, the sum value is given as base.

In EXP\_TO\_BASE state, input A, which is the base number, is given to ALUinA in order to perform the addition operation as A number.

In the MULT\_SUB state , we subtract 1 from input A as we did in case of EXP\_SUB.

In ADD\_REC, we assign the subtracted number A to register5. We decide how many times the addition operation will be performed on this number. The subtracted A number is still kept in the ALUout wire.

In the MULT\_ADD state, the subtracted A value in ALUout is checked to see if it is zero. If z = 1, the addition is finished. Go to EXP\_SUB state and subtract one from the exponential number (B) and continue. If not, the base value held in register6 is selected and given to ALUinA.

In SEL\_SUM\_A, the sum value held in register4 is given as feedback and transferred to ALUinB. We have given the base value to ALUinA. ADD block is selected with insel and the base value and total value are added.

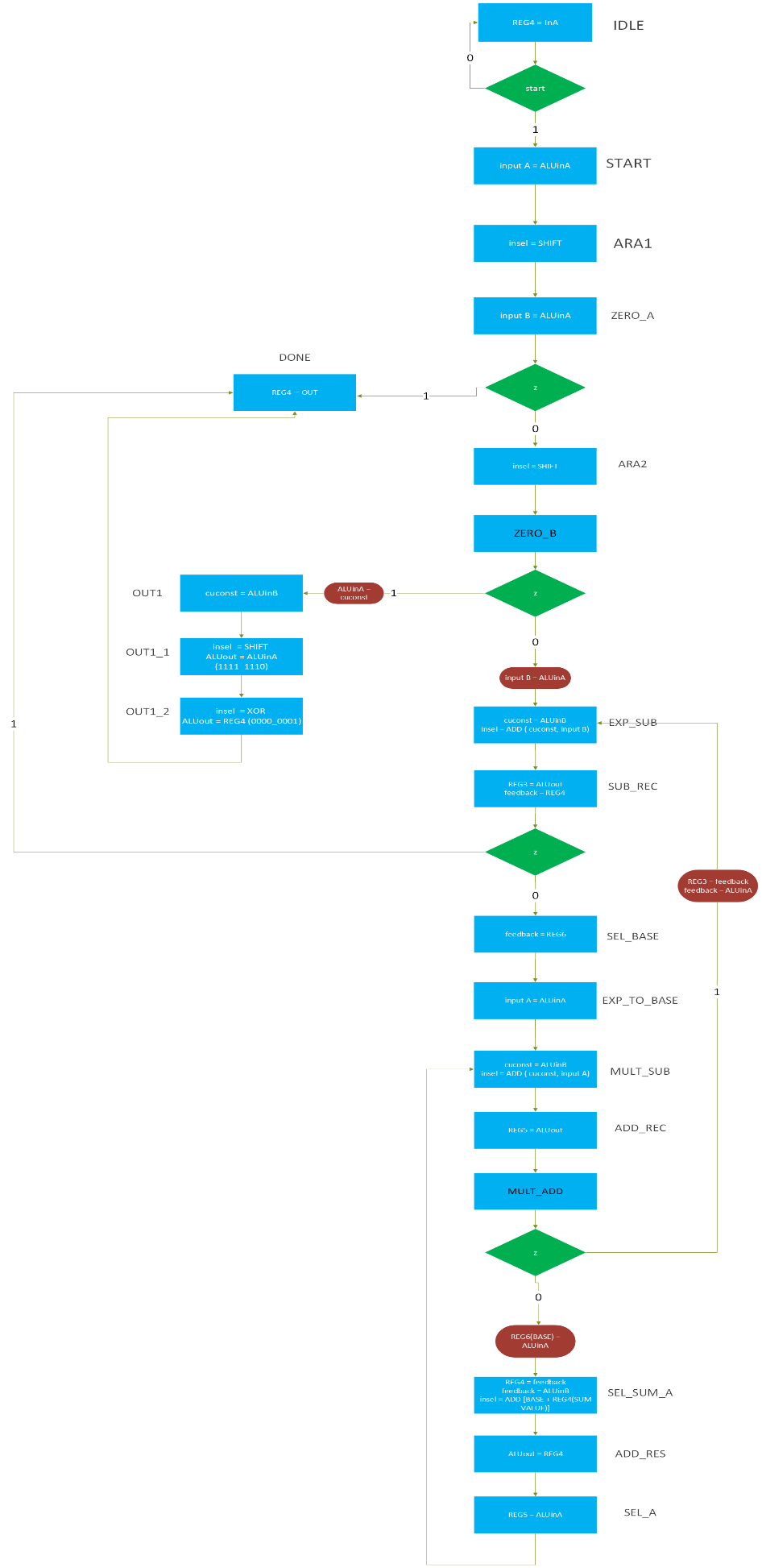
In the ADD\_RES state , the sum result is assigned to register4.

In the SEL\_A state, we give the value which is calculated in the ADD\_REC state and held in register5 to ALUinA. We return to the MULT\_SUB state.

metin, taslak, çizim, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şekil RTL Schematic of Control Unit

1. **Algorithmic State Machine**

1. **Top Module**

Şekil Verilog Code of Top Module

**module** top\_module

**(**

**input** clk**,**

**input** rst**,**

**input** **[**7**:**0**]** inA**,**

**input** **[**7**:**0**]** inB**,**

**input** start**,**

**output** busy**,**

**output** **[**7**:**0**]** out

**);**

**wire** co**;**

**wire** z**;**

**wire** **[**1**:**0**]** insel**;**

**wire** **[**7**:**0**]** cu\_const**;**

**wire** **[**2**:**0**]** in\_mux\_add**;**

**wire** **[**3**:**0**]** out\_mux\_add**;**

**wire** **[**3**:**0**]** reg\_add**;**

**wire** we**;**

**wire** **[**7**:**0**]** alu\_out**;**

**wire** **[**7**:**0**]** alu\_in\_a**;**

**wire** **[**7**:**0**]** alu\_in\_b**;**

**wire** **[**4**:**0**]** cu\_state **=** cu**.**state **;**

control\_unit cu

**(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**start**(**start**),**

**.**co**(**co**),**

**.**z**(**z**),**

**.**busy**(**busy**),**

**.**insel**(**insel**),**

**.**cu\_const**(**cu\_const**),**

**.**in\_mux\_add**(**in\_mux\_add**),**

**.**out\_mux\_add**(**out\_mux\_add**),**

**.**reg\_add**(**reg\_add**),**

**.**we**(**we**)**

**);**

**wire** **[**7**:**0**]** re\_out0 **=** rb**.**re\_out0**;**

**wire** **[**7**:**0**]** re\_out1 **=** rb**.**re\_out1**;**

**wire** **[**7**:**0**]** re\_out2 **=** rb**.**re\_out2**;**

**wire** **[**7**:**0**]** re\_out3 **=** rb**.**re\_out3**;**

**wire** **[**7**:**0**]** re\_out4 **=** rb**.**re\_out4**;**

**wire** **[**7**:**0**]** re\_out5 **=** rb**.**re\_out5**;**

**wire** **[**7**:**0**]** re\_out6 **=** rb**.**re\_out6**;**

**wire** **[**7**:**0**]** out\_mux\_out **=** rb**.**out\_mux\_out**;**

**wire** **[**7**:**0**]** in\_mux\_out **=** rb**.**in\_mux\_out**;**

REGISTER\_BLOCK rb

**(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**inA**(**inA**),**

**.**inB**(**inB**),**

**.**cu\_const**(**cu\_const**),**

**.**in\_mux\_add**(**in\_mux\_add**),**

**.**out\_mux\_add**(**out\_mux\_add**),**

**.**reg\_add**(**reg\_add**),**

**.**we**(**we**),**

**.**alu\_out**(**alu\_out**),**

**.**alu\_in\_a**(**alu\_in\_a**),**

**.**alu\_in\_b**(**alu\_in\_b**),**

**.**out**(**out**)**

**);**

**wire** **[**7**:**0**]** add\_sum **=** alu**.**add\_sum**;**

ALU alu

**(**

**.**insel**(**insel**),**

**.**alu\_in\_a**(**alu\_in\_a**),**

**.**alu\_in\_b**(**alu\_in\_b**),**

**.**alu\_out**(**alu\_out**),**

**.**co**(**co**),**

**.**z**(**z**)**

**);**

**endmodule**

We combined ALU, REGISTER BLOCK, CONTROL UNIT modules in the Top Module. I put additional wires to see the data in the Register Block and the states of the control unit.

**#**10**;**

inA **=** 8'd1**;**

inB **=** 8'd1**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd1**;**

inB **=** 8'd2**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd1**;**

inB **=** 8'd3**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd2**;**

inB **=** 8'd0**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd2**;**

inB **=** 8'd1**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd2**;**

inB **=** 8'd2**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd2**;**

inB **=** 8'd3**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**module** top\_module\_tb**;**

// Testbench sinyalleri

**reg** clk**;**

**reg** rst**;**

**reg** start**;**

**reg** **[**7**:**0**]** inA**;**

**reg** **[**7**:**0**]** inB**;**

**wire** busy**;**

**wire** **[**7**:**0**]** out**;**

// Test edilen modülün örneklenmesi

top\_module uut **(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**inA**(**inA**),**

**.**inB**(**inB**),**

**.**start**(**start**),**

**.**busy**(**busy**),**

**.**out**(**out**)**

**);**

// Clock üretimi

**always** **#**5 clk **=** **~**clk**;**

// Test senaryoları

**initial** **begin**

clk **=** 0**;**

rst **=** 1**;** // Reset aktif

start **=** 0**;**

// Reset işleminden çık

**#**10 rst **=** 0**;**

**#**10**;**

inA **=** 8'd0**;**

inB **=** 8'd1**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd0**;**

inB **=** 8'd2**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd0**;**

inB **=** 8'd3**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd1**;**

inB **=** 8'd0**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd1**;**

inB **=** 8'd1**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd1**;**

inB **=** 8'd2**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd1**;**

inB **=** 8'd3**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd2**;**

inB **=** 8'd0**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd2**;**

inB **=** 8'd1**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd2**;**

inB **=** 8'd2**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd2**;**

inB **=** 8'd3**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd3**;**

inB **=** 8'd0**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd3**;**

inB **=** 8'd1**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd3**;**

inB **=** 8'd2**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd3**;**

inB **=** 8'd3**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd7**;**

inB **=** 8'd2**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);**

**#**20

$display**(** "A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd3**;**

inB **=** 8'd5**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);**

**#**20

$display**(**" A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd6**;**

inB **=** 8'd3**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);**

**#**20

$display**(** "A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd5**;**

inB **=** 8'd3**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);**

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

// Testlerin sonu

**#**10**;**

$stop**;** // Simülasyonu durdur

**end**

**endmodule**

Şekil Simulation Codes of Top Module

**#**10**;**

inA **=** 8'd3**;**

inB **=** 8'd0**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd3**;**

inB **=** 8'd1**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd3**;**

inB **=** 8'd2**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd3**;**

inB **=** 8'd3**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);** // İşlem bitene kadar bekle

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd7**;**

inB **=** 8'd2**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);**

**#**20

$display**(** "A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd3**;**

inB **=** 8'd5**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);**

**#**20

$display**(**" A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd6**;**

inB **=** 8'd3**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);**

**#**20

$display**(** "A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

**#**10**;**

inA **=** 8'd5**;**

inB **=** 8'd3**;**

start **=** 1**;**

**#**10 start **=** 0**;**

**wait(**busy **==** 0**);**

**#**20

$display**(**"A=%d, B=%d, Out=%d"**,** inA**,** inB**,** out**);**

// Testlerin sonu

**#**10**;**

$stop**;** // Simülasyonu durdur

**end**

**endmodule**

ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**ekran görüntüsü, çizgi, renklilik içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Şekil Simulation Waveforms

In the job description, it was written that input A and input B would be 2 bits. Therefore, we tried all 2-bit inputs. In addition, we tried some values where the result of the operation would be at most 8 bits. Since the ports and wires are 8 bits, we could not try values that would produce more than 8 bits. When the busy signal is 0, the code gives the result to OUT. We got the correct result in all operations.

**metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şekil Console Outputs

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şekil Setup and Hold Delays

As can be seen in the tables, the maximum delay is 7.166 ns. Max clock frequency is 139.53 MHz.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şekil Utilization Report

94 LUT, 82 FF, 28 IO, 1 BUFG are used in the project

|  |  |  |
| --- | --- | --- |
| Name | Code | Report |
| Mehmet Yasir Bağcı | ALU  CONTROL UNIT | * Problem Statement * ALU * Register Block * ASM * Utilization Report |
| Salih Ömer Ongün | REGISTER BLOCK  CONTROL UNIT  TOP MODULE | * Control Unit * ASM * Top Module * Timing * Utilization Report |